UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO DE COMPUTADORES

LABORATÓRIO 27

FUNÇÕES INLINE E PONTEIROS PARA FUNÇÕES

EXERCÍCIOS DE REVISÃO

VOCÊ DEVE ACOMPANHAR PARA OBTER INFORMAÇÕES COMPLEMENTARES

1. Considerando os fragmentos de código abaixo:

```
double Multiplicar(double x, double y)
{
    return x * y;
}

void Atribuir(CharSet * dest, const char * texto)
{
    dest->tam = strlen(texto) + 1;

    // remove string antiga
    if (dest->str != nullptr)
        delete[] dest->str;

    // cria nova string
    dest->str = new char[dest->tam];

    // copia conteúdo para nova string
    int i = 0;
    while (texto[i])
        dest->str[i++] = texto[i];

    // finaliza string com \0
    dest->str[dest->tam - 1] = '\0';
}

int Tamanho(CharSet * texto)
{
    return texto->tam;
}

int Minimo(int a, int b)
{
    return (a < b ? a : b);
}
```

Quais funções são boas candidatas para serem inline?

2. Construa uma função para exibir mensagens de erro de um sistema. Algumas mensagens são direcionadas aos programadores, e neste caso elas devem ser apresentadas de forma simples, em apenas uma linha de texto.

```
Erro S2043: arquivo operation.sys não encontrado!
```

Outras mensagens são direcionadas aos usuários do sistema, e neste caso elas devem ser decoradas e exibidas em várias linhas de texto.

```
A operação não pôde ser concluída!

Algum problema foi detectado no sistema que impossibilitou a
conclusão da tarefa. Contacte o desenvolvedor do sistema em
dev@sys.com e envie o relatório de erros abaixo.

----------------------------
Relatório de Erros do Sistema
----------------------------
Código: S2043
Descrição: arquivo operation.sys não encontrado!
```

Ao invés de construir um if para selecionar a forma de exibição da mensagem, construa uma função ExibirErro que receba por parâmetro o número do erro, uma descrição do erro e uma função de exibição. Construa funções de exibição para refletir os dois cenários descritos anteriormente.

O programa principal deve ser semelhante ao código abaixo:

```
#include <iostream>
using namespace std;

// construa a função ExibirErro

// construa a função usuario
// construa a função programador
// ...


int main()
{
    ExibirErro(1245,
               "falha na leitura do arquivo",
               usuario);

    ExibirErro(4521,
               "erro inesperado na inicialização",
               programador);

    return 0;
}
```

EXERCÍCIOS DE FIXAÇÃO

VOCÊ DEVE FAZER OS EXERCÍCIOS PARA FIXAR O CONTEÚDO

1. Construa funções inline para as tarefas abaixo e escreva um programa para testar as funções. As funções devem ser criadas com apenas uma instrução em uma única linha.
    a. Achar o máximo entre dois números
    b. Achar o mínimo entre dois números
    c. Encontrar o valor absoluto de um número
    d. Converter um caractere para maiúsculo

2. Projete uma função `calcular()` que receba dois valores double e o endereço de uma função e retorne o resultado da aplicação dessa função sobre os valores recebidos. A função passada como argumento de `calcular()` deve receber dois argumentos tipo double e retornar um valor double. Por exemplo, suponha a definição:

```
double soma(double x, double y)
{
    return x * y;
}
```

Então a chamada de função abaixo deve fazer `calcular()` passar os valores `2.5` e `10.3` para a função `soma()` e retornar o resultado de soma, que é `12.8` neste exemplo.

```
double q = calcular(2.5, 10.3, soma);
```

Utilize estas funções e pelo menos mais uma função semelhante a `soma()` no programa. O programa deve usar um laço que permita ao usuário digitar pares de valores. Para cada par, use `calcular()` para chamar `soma()` e outra função.

3. No programa anterior use um vetor de ponteiros para funções do estilo de `soma()` e use um laço para sucessivamente aplicar `calcular()` em uma série de funções.

Dica: Aqui está a forma de declarar o vetor de ponteiros:

```
double (*pf[3]) (double, double);
```

É possível inicializar o vetor usando a notação de inicialização de vetores com os nomes das funções.

EXERCÍCIOS DE APRENDIZAGEM

VOCÊ DEVE ESCREVER PROGRAMAS PARA REALMENTE APRENDER

1. Ponteiros pra funções podem ser usados para simplificar códigos que devem escolher uma dentre várias funções com base no valor de uma variável. Por exemplo, considere o trecho de código abaixo:

```
switch(num)
{
case 0:
       func0();
       break;
case 1:
       func1();
       break;
case 2:
       func2();
       break;
case 3:
       func3();
       break;
}
```

Uma alternativa para o código acima é criar um vetor de ponteiros para funções, de forma que o valor da variável `num` seja usado para chamar a função adequada:

```
void (*func[])(void) = { func0, func1, func2, func3 };

...

func[num]();
```

Construa um programa que use um vetor de ponteiros para funções para implementar a seleção de uma opção de Menu. Cada vez que o usuário fizer uma escolha, uma mensagem apropriada deve ser exibida por uma função.

```
Menu do Sistema

1) Inserir
2) Remover
3) Buscar
4) Sair

Escolha:
```

2. Muitas bibliotecas que implementam funcionalidades do sistema operacional usam o conceito de função call-back. A função call-back é uma função que não é chamada diretamente pelo seu código, mas sim pelo código da biblioteca, ou do sistema onde está rodando.

   Por exemplo, para saber todos os dispositivos controladores de jogos que estão conectados em um computador, a biblioteca do sistema disponibiliza uma função `Enumerate` que, ao invés de retornar os dados dos dispositivos, chama uma função (função call-back) cada vez que encontra um dispositivo. Vamos construir um programa para simular este cenário. O registro abaixo mantém os dados de um dispositivo controlador de jogo.

```
struct Controller
{
    char name[40];
    int buttons;
    int axis;
};
```

   Construa duas funções call-back chamadas ListarNomes e ListarEixos, para exibir respectivamente os nomes dos controladores e as quantidades de eixos. As funções devem funcionar com o código abaixo:

```
#include <iostream>
using namespace std;

struct Controller
{
    char name[40];
    int buttons;
    int axis;
};

// --------------------------------
// crie aqui as funções para listar
// --------------------------------

void Enumerate(void(*f)(Controller))
{
    Controller vet[] =
    {
        {"Joy", 8, 4},
        {"Xbox", 10, 3},
        {"Play", 8, 6}
    };

    for (auto i : vet)
        f(i);
}

int main()
{
    Enumerate(ListarNomes);
    Enumerate(ListarEixos);
}
```

3. Funções do tipo call-back também são muito empregadas na criação de interfaces com o usuário (janelas, botões, caixas de texto, etc.). Considere uma função CreateButton que receba a posição x, y em que o botão deve ser desenhado dentro da janela e uma função a ser executada quando o botão for pressionado. O código abaixo está tentando simular esta situação.

   Complete o código criando a função CreateButton. Ela deve exibir uma mensagem informando a posição da tela em que o botão foi criado e atribuir a função recebida para o ponteiro global OnClick de forma que a saída seja como a seguir:

```
Botão criado na posição 10,10
Pressionar Botão? s
Botão Pressionado!
```

```
#include <iostream>
using namespace std;

void (*OnClick)(void);

// --------------------------------
// crie aqui a função CreateButton
// --------------------------------

void Mensagem()
{
    cout << "Botão Pressionado!" << endl;
}

int main()
{
    CreateButton(10, 10, Mensagem);

    // simulando pressionamento
    cout << "Pressionar Botão? ";
    char resposta;
    cin >> resposta;

    if (resposta == 'S' || resposta == 's')
        OnClick();

    return 0;
}
```